EE 301 Lab 7 – Filters in the Frequency Domain

In this lab, we will be using the frequency response of filters to examine its use in telephone touchtone dialing. In touchtone dialing, we are interested in reliably transmitting and detecting phone numbers across a noisy audio channel. AT&T's solution involved the transmission of a sum of sinusoids with particular frequencies.

1 What you will learn

In this lab we will be examining the frequency response of FIR filters. We will also be looking at dual tone multi-frequency signals, generating DTMF signals, bandpass filters and a DTMF decoder.

2 Background Information and Notes

2.1 DTMF signals and Touch Tone™ Dialing

When a particular number is pressed on a telephone touch pad, a unique tone is generated. Each tone is a sum of two sinusoids, and the resulting signal is known as a dual-tone multifrequency (or DTMF) signal. Table 7.1 shows the frequencies generated for each number/button. For example, the "6" button will generate a signal which is the sum of a 1336 Hz and a 770 Hz sinusoid.

Frequencies	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

Table 7.1: DTMF encoding table for touch tone dialing. When any button is pressed, the tones of the corresponding row and column are generated.

The set of the seven frequencies listed on Table 7.1 is called the *DTMF frequencies*. One can see that none of the DTMF frequencies is a multiple of another and so chosen to minimize the effects of signal distortions.

The DTMF signal in the time domain does not convey much information, but by using a spectrogram we can glean more useful information of the DTMF signal. The *spectrogram* allows one to see the frequency properties of a signal as they change over time. The spectrogram involves taking multiple Discrete Fourier Transforms (DFTs) over small, overlapping segments of a signal. The resulting DFT magnitudes are then combined in a matrix and displayed as an image. A spectrogram of a DTMF signal is shown in Figure 7.1. The x-axis represents time and the y-axis represents frequency. The bars represent a sinusoid of a particular frequency existing over some time period. Notice that at each time, there are two bars, which show the presence of the two sinusoids that make up the DTMF tone. From this one can identify the number that was pressed.

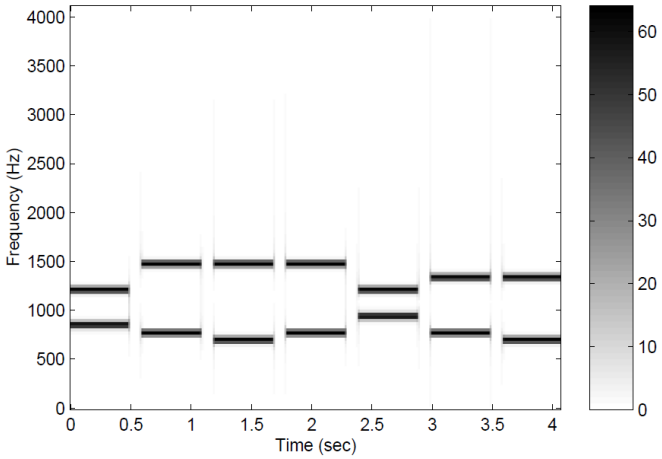

Figure 7.1: A spectrogram of a DTMF signal. Each horizontal bar indicates a sinusoid that exists over some time period.

2.2 Decoding DTMF Signals

To decode DTMF signals requires several steps. The first two steps will determine the strength of the signal at each of the DTMF frequencies. The signal passes through a set of bandpass filters with center frequencies at each of the DTMF frequencies. The output of the filters will indicate the signal strength. The third step involves detecting and decoding. From the filter output strengths, we detect whether or not a DTMF signal is present. If signal is present, we select the two filters with the largest output strengths and determine the button that was pressed. A block diagram of the DTMF decoder is shown in Figure 7.2.

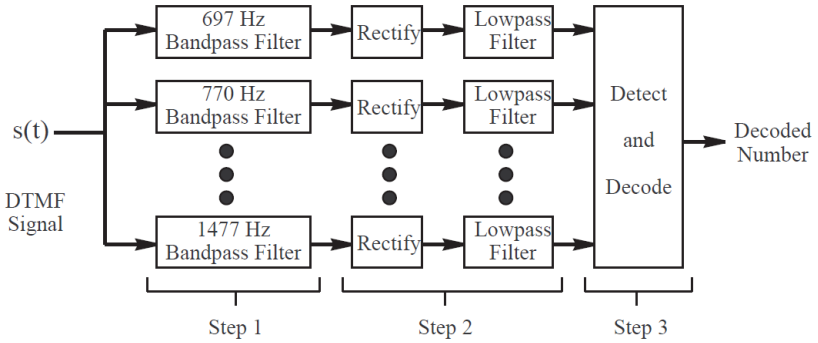

Figure 7.2: A block diagram of the DTMF decoder system. The input is a DTMF signal, and the output is a string of numbers corresponding to the original signal.

Step 1: Bandpass Filters

From previous labs, you may remember that correlating two signals involves measuring how similar those two signals are. Since convolution is similar to correlation except with a time reversal, we can use this same idea to design a filter that passes a given frequency. If our filter's impulse response looks similar to the input signal, we will get a large amplitude output signal. Otherwise, if it is not similar, it will be a smaller amplitude output signal.

The type of filter we are interested in using is a bandpass filter. The filter's *center frequency* represents the frequency that we wish to pass. Thus for a bandpass filter with center frequency f_c, we want our impulse response, h, to be

$$h[k] = \begin{cases} \sin(2\pi f_c k / f_s) & 0 \le k \le M \\ 0 & \text{else} \end{cases} \tag{7.1}$$

The above equation is a FIR filter with order M. (Note that the support length of the impulse response is M + 1.) M is a design parameter, but there is a tradeoff. If M is large, one can obtain a better differentiation between passed frequencies and rejected frequencies, however more computation is required to perform the convolution.

Since we have seven DTMF frequencies, we will need seven bandpass filters in our system. Each bandpass filter will have a different value of M.

In order to know how good a bandpass filter is at rejecting unwanted DTMF frequencies, we will define the *gain-ratio*, R, which is,

$$R = \frac{|\mathrm{H}(f_c)|}{\max|\mathrm{H}(\hat{f})|} \tag{7.2}$$

where f_c is the center frequency, H is the frequency response and $\hat{f}$ is in the set of DTMF frequencies, where $\hat{f} \neq f_c$. R is defined to be the ratio of the filter's gain at its center frequency to the *next-highest* gain at one of the DTMF frequencies. It is desirable to have a high gain-ratio because it indicates that the strength at which the filter is rejecting the other possible frequencies.

Since we need to compare the outputs of a variety of bandpass filters, one need to normalize each filter by the center frequency gain. The M value and the center frequency gain will need to be recorded.

Step 2: Determining filter output strengths

In order to measure the strength of the filter's output, we need to measure the envelope of the filter outputs. To just follow the signal's positive envelope of the signal, we can eliminate the negative portion of the signal. A *half-wave rectifier* will eliminate all parts of the signal below zero. Alternatively , a *full-wave rectifier* will take the absolute value of the signal. In practice, half-wave rectifiers are easier to design. However, full-wave rectifiers are preferable, and easy to implement in MATLAB. Thus, for this lab we will use full-wave rectifiers. Figure 7.3 shows the output of these two types of rectifiers.

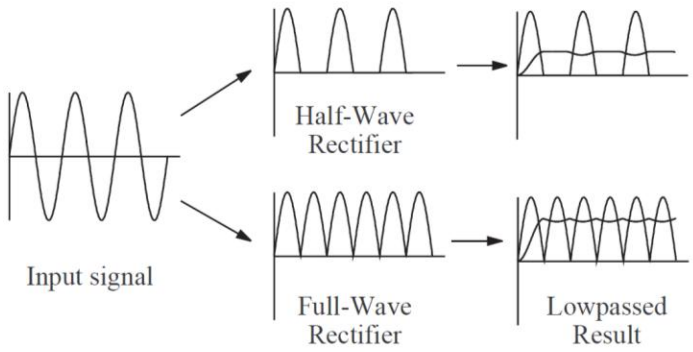

Figure 7.3: A comparison of half-wave and full-wave rectification. Notice that full-wave rectification allows us to achieve a higher output signal level after lowpass filtering.

The rectified signal is then passed through a smoothing filter. The output will be a nearly constant signal that indicates the strength of the filter's input at the center frequency of the filter. This smoothing will be performed by using a simple moving average filter with the following impulse response,

$$h_{LP} = \begin{cases} \dfrac{1}{M_{LP}+1} & 0 \le k \le M_{LP} \\ 0 & \text{else} \end{cases} \tag{7.3}$$

The filter order is M_{LP}. The value M_{LP} (and subsequently the corresponding strength of the smoothing filter) is a design parameter, and there is a tradeoff between the amount of smoothing and transient effects. If the filter's impulse response is not long enough, the output signal will continue to have significant variations. If the filter's impulse response is too long, transient effects will dominate in the output. Also, if it is too short, the system may smooth over short DTMF tones or periods of silence. In our decoder system, the same smoothing filter is applied to the output of each filter. The results of smoothing for half-wave and full-wave rectified signals is shown in Figure 7.3.

Step 3: Detect and Decode

Once the outputs of the bandpass filters are processed, we are now ready to detect whether or not a DTMF tone is present or not, and, if it is, determine which button was pressed to generate it.

The first step is to detect whether a DTMF tone is actually present at a particular time. To detect the presence of a DTMF tone one can compare the rectified and smoothed bandpass filter outputs, to a threshold, c. If a signal is greater than the threshold, then we decide that a DTMF tone is present. The threshold should be high enough that noise will not trigger the detector during a period of silence, but low enough that noise won't pull the signal below the threshold.

When the input signal is noisy, there is another problem during the transient portions at the beginning and end of a DTMF tone. Near the threshold crossing, the noise could cause the signal to cross the threshold several times and could cause a single DTMF tone to be decoded as

multiple button presses. To remedy this problem, we could only make a decision every 100 samples.

The second step is to decode or decide which button was pressed to generate a particular DTMF tone that we detected in the previous step. We first determine which two bandpass filters have the largest output at each time when a DTMF tone was detected. Then we have a look-up table to see which button was pressed.

The third step simply combines adjacent, identical numbers in the decoded sequence. A sequence of identical numbers is replaced by a single number. However for this process to work correctly, an detection of when no tone is present is also needed. Otherwise, any repeated button press would be decoded as only a single button press.

2.3 Searching Parameter Spaces

The basic idea in looking for a "good value" of a design parameter is that we want to get in the ballpark before we concern ourselves with the optimum solution. First think about varying the parameter over factors of 2 or factors of 10. For example you could try 0.01, 0.1, 1, 10, and 100 to get a general understanding of how the system responds to the varying parameter. Once that is understood, a smaller range could then be analyzed to obtain the optimum value.

2.4 Some MATLAB commands for this lab

- **Computing the frequency response of an FIR filter**: The MATLAB command **freqz** returns the frequency response of a filter at a specified number of discrete-time frequencies. The general usage of **freqz** for causal FIR filters is:

  >> [H,w] = **freqz**(bb,1,n);

  Here, bb is the set of filter coefficients (i.e., the impulse response) of the FIR filter, n is the number of points in the range [0; _) at which to evaluate the frequency response, H is the frequency response, and w is the set of n corresponding discrete-time frequencies, which are spaced uniformly from 0 to _. The frequency response, H, is a vector of complex numbers which define the *gain* (abs(H)) and *phase-shift* (angle(H)) of the filter at the given frequencies.

  Alternatively, we can evaluate the frequency response only at a specified set of frequencies by replacing with a vector of discrete-time frequencies. Thus, the command

  >> H = **freqz**(bb,1,[pi/3, pi/2, 2*pi/3]);

  returns the frequency response at the discrete-time frequencies $\pi/3$, $\pi/2$ and $2\pi/3$.
  When we apply a filter to a sampled signal with sampling frequency fs (in samples per second), we can evaluate the frequency response at the discrete-time frequencies corresponding to a specified set of continuous time frequencies in Hertz in the following manner:

  >> H = **freqz**(bb,1,[100 200 400 500]/fs*2*pi);

  This converts the specified continuous-time frequencies into discrete-time frequencies and evaluates the frequency response at those points.

- **Creating matrices of ones and zeros:** In order to create arrays of arbitrary size containing only ones or only zeros, we use the MATLAB ones and zeros commands. Both commands take the same

set of input parameters. If only one input parameter is used, a square matrix with the specified number of rows and columns is generated. For instance, the command

```
>> x = ones(5);
```

produces a 5 x 5 matrix of ones. Two parameters specify the desired number of rows and columns in the matrix. For instance, the command

```
>> x = zeros(4, 8);
```

produces a 4 x 8 matrix (i.e., four rows and eight columns) containing only zeros. To generate column vectors or row vectors, we set the first or second parameter to 1, respectively.

- **The DTMF Dialer:** dtmf_dial.m is a DTMF "dialer" function. It takes a vector of button presses (i.e., a phone number) and produces the corresponding audio DTMF signal. *Note that this function as provided is incomplete; you will be directed to complete it in the laboratory assignment.* (The lines of code that you need to complete are marked with a ?.) To produce the DTMF signal that lets you dial the number 555-2198, use the command:

  ```
  >> signal = dtmf_dial([5 5 5 2 1 9 8]);
  ```

  An optional second parameter will cause the function to display a spectrogram of the resulting DTMF signal:

  ```
  >> signal = dtmf_dial([5 5 5 2 1 9 8],1);
  ```

  This function assumes a sampling frequency of 8192 samples per second. Each DTMF tone has a length of 1/2 second, and the tones are separated by 1/10 second of silence. Note that the number 10 corresponds to a '#', 11 corresponds to a '0', and 12 corresponds to a '*'.

- **The DTMF Decoder:** dtmf_decode.m is an (incomplete) DTMF decoder function. (Once again, the lines of code that you need to complete are marked with a ?.) It takes a DTMF signal (as generated by dtmf_dial) and returns the sequence of button-presses used to create the signal. Thus, if our DTMF signal is stored in signal, we decode the signal using the command:

  ```
  >> decoded = dtmf_decode(signal);
  ```

  An optional second parameter will cause the function to display a plot of the smoothed and rectified outputs of each bandpass filter:

  ```
  >> decoded = dtmf_decode(signal,1);
  ```

- **Bandpass Filter Characterization:** dtfm_filt_char.m is a function that we will use to help us calculate gain-ratios for the bandpass filters used in the DTMF decoder. We use the function to focus on one of the bandpass filters at a time. The function takes two parameters: the order, M, of one of the bandpass filter's impulse responses and the center frequency in Hertz, frq, of that filter. The function returns a vector containing the gain (i.e., the magnitude of the frequency response) at each of the DTMF frequencies, from lowest to highest. It also produces a plot of the frequency response with locations of the DTMF frequencies indicated. Use the following command to execute the function:

  ```
  >> gains = dtmf_filt_char(M,frq);
  ```

A second optional parameter lets you suppress the plot:

```
>> gains = dtmf_filt_char(M,frq,0);
```

- **Testing the robustness of the DTMF decoder:** dtmf_attack.m is a function that tests the DTMF decoder in the presence of random noise. This function generates a standard seven digit DTMF signal, adds a specified amount of noise to the signal, and then passes it through your completed dtmf_decode function. The decoded string of button presses is compared to those that generated the signal. Since the noise is random, this procedure is repeated ten times. The function then outputs the fraction of trials decoded successfully. The function also displays the plot from the last execution of dtmf_decode. (Note: since each call to dtmf_decode takes a little time, this function is rather slow. Be patient with it.)

  For instance, to test the system with a noise power of 2.5, we use the following command:

  ```
  >> success_rate = dtmf_attack(2.5);
  ```

  The result is a number that provides the fraction of the 10 trials that were successful. Although dtmf_attack is a complete function, it calls dtmf_dial and dtmf_decode, each of which you must complete.

3 Guided Exercises

1. (The DTMF dialer.) Before we can decode a DTMF signal, we need to be able to produce DTMF signals. In this problem, we'll write a function that takes a phone number and produces the corresponding DTMF signal, just like the telephone would produce if you dial the number.

   Download the function **dtmf_dial.m**, which is a nearly complete dialer function. You simply need to replace the question marks by code that completes the function. The first missing line of code generates a DTMF tone for each number in the input and appends it to the output signal. The second line of code appends a short silence to the signal to separate adjacent DTMF tones.
   - Complete the function and include the code in your lab report.
   - Using your newly completed dialer function, execute the following command to create a DTMF signal and display it's spectrogram:

     ```
     >> signal = dtmf_dial([1 2 3 4 5 6 7 8 9 10 11 12],1);
     ```

     Include the resulting figure in your report. Note how each button press produces a different pattern on the spectrogram.
   - What is the phone number that has been dialed in Figure 7.1?

2. (The bandpass filters of the DTMF Decoder.) As we have noted, a key part of the DTMF decoder is the set of bandpass filters that is used to detect the presence of sinusoids at the DTMF frequencies. We have specified a general form for the bandpass filters, but we still need to choose the filter orders and create their impulse responses. In this problem you will be identifying good values for M.

   (a) (The impulse response of one bandpass filter.) First, we need to be able to create the impulse response for a bandpass filter. Using equation (7.1) with a sampling frequency f_s = 8192 Hz and M

= 50, use MATLAB to create a vector containing the impulse response, h, of a 770 Hz bandpass filter.

- What is the command that you used to create this impulse response?
- Use stem to plot your impulse response.

(b) (The frequency response of one bandpass filter.) When we talk about the response of a filter to a particular frequency, we can think about filtering a unit amplitude sinusoid with that frequency and measuring the amplitude and phase shift of the resulting signal. We can certainly do this in MATLAB, but it's far simpler to use the **freqz** command. Here, you'll use **freqz** to examine the frequency response and gain-ratio of a bandpass filter like the ones we'll use in the DTMF decoder.

- Use **freqz** to calculate the frequency response of your 770 Hz bandpass filter at all seven of the DTMF frequencies[4]. Calculate the gain at each frequency, and include these numbers in your report.
- From the frequency response of your filter at these frequencies, calculate the gain-ratio, R.

(c) (Choosing M for this bandpass filter.) Now, we'd like to see what happens when we change M for your 770 Hz bandpass filter. We've provided you with a function that will facilitate this. Download the file **dtmf_filt_char.m**. This function will help you to visualize the frequency response of these filters and to determine their gain at the DTMF frequencies.

- Use this function to verify that the gains you calculated in Problem 2b were correct.
- Include the frequency-response plot that **dtmf_filt_char** produces in your report.
- The frequency response of this filter is characterized by several "humps" which are typically called *lobes*. Describe the frequency response in terms of such lobes. Vary M and examine the plots that result (you do not need to include these plots). Describe the differences in the frequency response as M (which represents the length of the filter's impulse response) is changed.
- What happens to the relative heights of adjacent lobes as M is changed?
- What features of the filter's frequency response contribute to the gain ratio R?
- For what values of M do we achieve gain ratios greater than 10?

(d) (A function for computing gain ratios.) You'll need to compute the gain-ratio repeatedly while finding good design parameters for the bandpass filters, so in this problem you'll automate this task. Write a function that accepts a vector of gains (such as that returned by **dtmf_filt_char**) and computes the gain ratio, R. (Hint: This is a simple function if you use the sort command. You can assume that the center frequency gain is the largest value in the vector of gains.)

- Include the code for this function in your report.

(e) (Specifying the bandpass filters.) For each bandpass filter that corresponds to one of the seven DTMF frequencies, we want to find a choice of M that yields a good gain ratio but also minimizes the computation required for filtering.

To do this, for each bandpass filter frequency, use **dtmf_filt_char** and your function from Problem 2d to calculate R for all M between 1 and 200. Then, plot R as a function of M. You can save some computation time by setting the third parameter of **dtmf_filt_char** to zero to suppress plotting. You should be able to identify at least one local maximum[5] of R on the plot. The "optimal" value of M that we are looking for is the smallest one that produces a local maximum of R that is greater than 10.

- Create this plot of R as a function of M for the bandpass filter with a center frequency of 770 Hz. Include the resulting plot in your report.
- Identify the "optimal" value of M for this filter, the associated center frequency gain, and the resulting value of R.

- Repeat the above two steps for the remaining six bandpass filters. (You do not need to include the additional plots in your report.) Create a table in which you record the center frequency, the optimal M value, the associated center frequency gain, and the resulting value of R.

3. (Completing the DTMF decoder.) Now we have designed the set of bandpass filters that we need for the DTMF decoder. In this problem, we'll use the parameters that we found to help us complete the decoder design.

   Download the file **dtmf_decode.m**. This function is a nearly complete implementation of the DTMF decoder system described earlier in this lab. There are several things that you need to add to the function.

   (a) (Setting the M's and the gains of the bandpass filters.) First, you need to record your "optimized" values of M and the center frequency gains in the function. Replace the question marks on by a vector of your optimized values of M. They should be in order from smallest frequency to largest frequency. Do the same for the variable G, which contains the center frequency gains.
   - Make these modifications to the code. (At the end of this problem, make sure that you include your completed function in your report.)

   (b) (Setting the impulse responses of the bandpass filters.) Also, you need to define the impulse response for each bandpass filter. Use equation (7.1) for this, where the filter's order is given by M(i).
   - Make this modification to the code.

   (c) (Selecting the order of the post-rectifier smoothing filter.) Next, you need to specify the post-rectifier smoothing filter, **h_smooth**. Temporarily set both **h_smooth** and threshold equal to 1 and run **dtmf_decode** on the DTMF signal you generated in Problem 1. This function displays a figure containing the rectified and smoothed outputs for each bandpass filter. With **h_smooth** equal to 1, no smoothing is done and we only see the results of the rectifier in this figure. We will use moving average filters of order M_{LP}, as defined by the MATLAB command

   ```
   >> h_smooth = ones(M_LP+1,1)/(M_LP+1);
   ```

   We want the smoothed output to be effectively constant during most of the duration of the DTMF tones, but we don't want to smooth so much that we might miss short DTMF tones or pauses between tones.
   - Examine the behavior of the smoothed signal when you replace with moving average filters with order M_{LP} equal to 20, 200, and 2000. Which filter order, M_{LP} gives us the best tradeoff between transient effects and smoothing?
   - Set **h_smooth** to be the filter you have just selected.

   (d) (Detection threshold.) Finally, you need to identify a good value for **threshold**. **threshold** determines when our system detects the presence of a DTMF signal. **dtmf_decode** plots the threshold on its figure as a black dotted line. We want the threshold to be smaller than the large amplitude signals during the steady-state portions of a DTMF signal, but larger than the signals during the start-up transients for each DTMF tone. (Hint: When choosing a threshold, consider what might happen if we add noise to the input signal.)
   - By looking at the figure produced by **dtmf_decode**, what would be a reasonable threshold value? Why did you choose this value?
   - Set **threshold** to the value you have just selected.

- Now, execute **dtmf_decode** and include the resulting plot in your report. (Note: You can include this plot in black and white, if you like.)
- **dtmf_decode** should output the same vector of "button presses" that was used to produce your signal. What "button presses" does the function produce? Do these match the ones used to generate the DTMF signal? If not, you've probably made a poor choice of threshold.

(e) Remember to include the code for your completed **dtmf_decode** function in your report.

4. (Robustness of the DTMF decoder to noise.) In the introduction to this lab, we indicated that we would be transmitting our DTMF signals over a noisy audio channel. So far, though, we have assumed that the decoder sees a perfect DTMF signal. In this problem, we will examine the effects of additive noise on the DTMF decoder.

   Download the file **dtmf_attack.m**. Execute **dtmf_attack** with various noise powers. Find a value of noise power for which some but not all of the trials fail.
   - What value of noise power did you find? (Hint: use the parameter searching method discussed in the background section to speed your search).
   - Make a plot of the fraction of successes versus noise power. Include at least 10 values on your plot. Make sure that your minimum noise power has a success rate at (or at least near) 1 and your maximum noise power has a success rate at (or near) 0. Try to get a good plot of the transition between high success rates and low success rates. While making this plot, pay attention to the types of errors that the decoder is making.

4 Review

1. None.

5 Lab Report

1. The first page of your Lab report should be a cover sheet with your name, USC ID and Lab #. Please note that all reports should be typed.
2. Answer all the questions which were asked in the lab report. Kindly display the code lines you executed to arrive at your answer along with figures to support them. Please give written explanation or put comment lines where necessary. Please note that each figure should have proper labels for the x and y axis and should have a suitable title.
3. Answer the review questions.
4. Submit a printout of your completed M-file documenting all the lab exercises.

This lab document and the figures contained were adapted from a University of Michigan Signals and Systems course lab handout (2002).